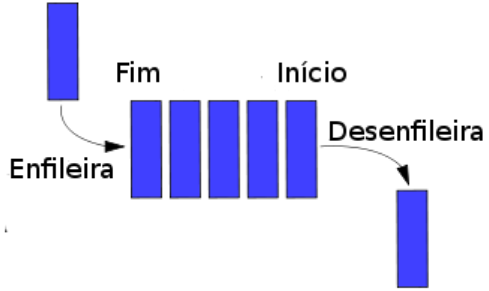
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

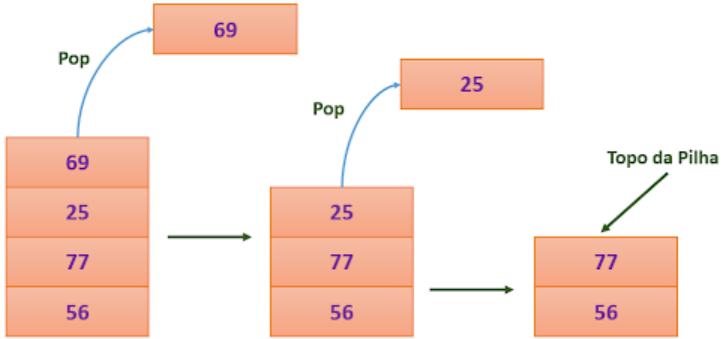
Algoritmos e Estruturas de Dados I

PROF. EDÉCIO FERNANDO IEPSEN

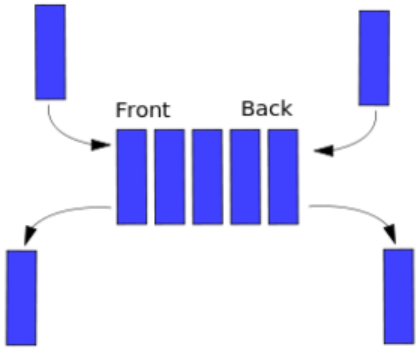
Filas



Pilhas



Deques (*Double Ended Queue*)



4.1 Fundamentos

Fila é uma lista em que as *inserções* são feitas num extremo, denominado *final*, e as *remoções* são feitas no extremo oposto, denominado *início*.

Quando um novo item é inserido numa fila, ele é colocado em seu final e, em qualquer instante, apenas o item no início da fila pode ser removido. Devido a essa política de acesso, os itens de uma fila são removidos na *mesma ordem* em que foram inseridos, ou seja, o primeiro a entrar é o primeiro a sair (Figura 4.1). Por isso, as filas também são denominadas listas FIFO (*First-In/First-Out*).

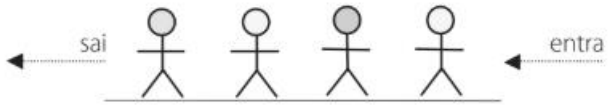


Figura 4.1 | Uma fila de pessoas: a primeira que entra é a primeira que sai.

A principal propriedade de uma fila é a sua capacidade de *manter a ordem* de uma sequência. Essa propriedade é útil em várias aplicações em computação.

Por exemplo, em um sistema operacional, cada solicitação de impressão de documento feita pelo usuário é inserida no final de uma fila de impressão. Então, quando a impressora fica livre, o gerenciador de impressão atende à próxima solicitação de impressão, removendo-a do início dessa fila. Assim, as solicitações de impressão são atendidas na mesma ordem em que elas são feitas.

Uma fila também é usada num sistema operacional para gerenciar a entrada de dados via teclado. À medida que as teclas são pressionadas pelo usuário, os caracteres correspondentes são inseridos numa área de memória chamada *buffer* de teclado. Então, quando um caractere é lido por um programa, por exemplo, com a função `getchar()`, declarada em `stdio.h`, o primeiro caractere inserido no *buffer* de teclado é removido e devolvido como resposta. Assim, os caracteres são processados na mesma ordem em que são digitados pelo usuário.

4.2 Operações em filas

Uma fila `F` suporta as seguintes operações:

- `fila(m)`: cria e devolve uma fila vazia `F`, com capacidade máxima `m`.
- `vaziaf(F)`: devolve `1` (*verdade*) se `F` está vazia; senão, devolve `0` (*falso*).
- `cheiaf(F)`: devolve `1` (*verdade*) se `F` está cheia; senão, devolve `0` (*falso*).
- `enfileira(x,F)`: insere o item `x` no final da fila `F`.
- `desenfileira(F)`: remove e devolve o item existente no início da fila `F`.
- `destroif(&F)`: destrói a fila `F`.

2.1 Fundamentos

Pilha é uma lista em que todas as operações de inserção, remoção e acesso são feitas num mesmo extremo, denominado *topo*.

Quando um item é inserido numa pilha, ele é colocado em seu topo e, em qualquer instante, apenas o item no topo da pilha pode ser removido. Devido a essa política de acesso, os itens são removidos da pilha na *ordem inversa* àquela em que foram inseridos, ou seja, o último a entrar é o primeiro a sair (Figura 2.1). Por isso, pilhas são também denominadas listas LIFO (*Last-In/First-Out*).

Figura 2.1 | Uma pilha de livros: o último livro empilhado é o primeiro a ser desempilhado.

A principal propriedade de uma pilha é a sua capacidade de *inverter a ordem* de uma sequência. Essa propriedade é útil em várias aplicações em computação.

Por exemplo, num navegador *web*, conforme as páginas vão sendo acessadas, seus endereços vão sendo inseridos numa pilha. Em qualquer instante durante a navegação, o endereço da última página acessada está no topo da pilha. Quando o botão *voltar* é clicado, o navegador remove um endereço da pilha e recarrega a página correspondente. Então, à medida que o botão *voltar* é clicado, as páginas acessadas são reapresentadas na ordem inversa àquela em que foram visitadas.

Controle do fluxo de execução é outro exemplo interessante do uso de pilha. Durante a execução de um programa, sempre que uma função é chamada, antes de passar o controle a ela, um endereço de retorno correspondente é inserido numa pilha. Quando a função termina sua execução, o endereço no topo da pilha é removido e a execução do programa continua a partir dele. Assim, a última função que passa o controle é a primeira a recebê-lo de volta (Figura 2.2).

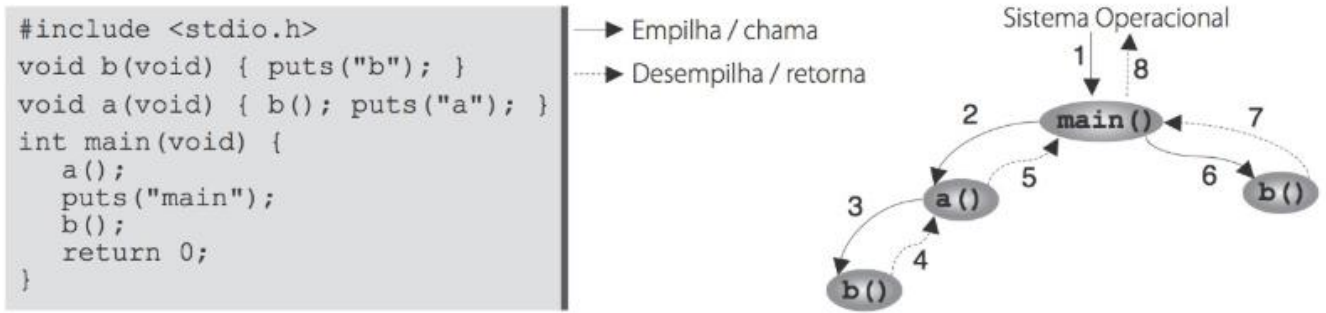


Figura 2.2 | Fluxo de execução: a ordem de retornos é inversa à ordem de chamadas.

2.2 Operações em pilhas

Uma pilha `P` suporta as seguintes operações:

- `pilha(m)`: cria e devolve uma pilha vazia `P`, com capacidade máxima `m`;
- `vaziap(P)`: devolve `1` (*verdade*) se `P` está vazia; senão, devolve `0` (*falso*);
- `cheiap(P)`: devolve `1` (*verdade*) se `P` está cheia; senão, devolve `0` (*falso*);
- `empilha(x,P)`: insere o item `x` no topo da pilha `P`;
- `desempilha(P)`: remove e devolve o item existente no topo da pilha `P`;
- `topo(P)`: acessa e devolve o item existente no topo da pilha `P`;
- `destroip(&P)`: destrói a pilha `P`.

4.3 → fila dupla – *deque*

Uma fila dupla, também conhecida como "deque", é um tipo especial de fila, na qual é permitido o acesso a qualquer uma das duas extremidades da lista, mas somente às extremidades (Figura 4.8). Inserções, alterações, remoções e consultas podem ser realizadas tanto no início quanto no final da fila dupla. Exemplos deste tipo de lista são:

- canais de navegação marítima ou fluvial;
- servidões com circulação nos dois sentidos.

As seguintes operações podem ser realizadas sobre filas duplas:

- criar a fila dupla vazia;
- inserir um novo nodo em uma das duas extremidades;
- excluir o nodo que está em uma das duas extremidades;
- consultar e/ou modificar o nodo que está em uma das duas extremidades;
- destruir a fila, liberando o espaço que estava reservado para ela.

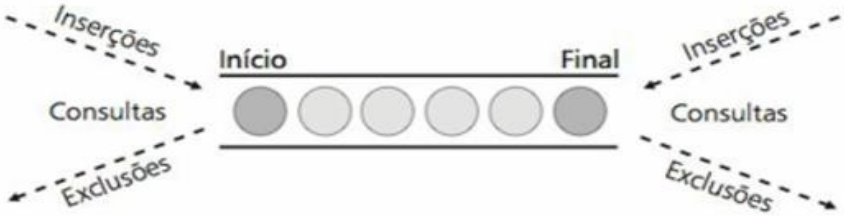
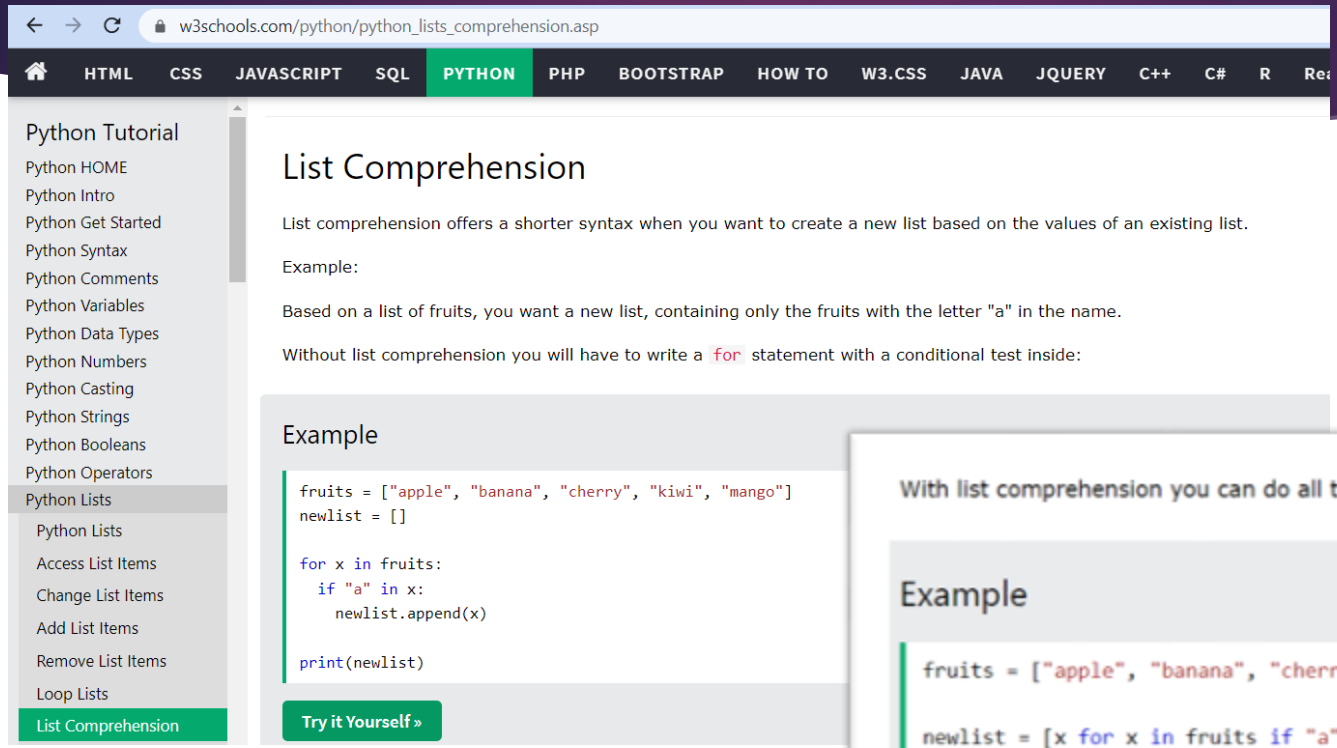


Figura 4.8 Fila dupla (*Deque*).

Estruturas de Dados

EDELWEISS,Nina ; GALANTE, Renata

List Comprehension



With list comprehension you can do all that with only one line of code:

Example

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)
```

Exemplos: List Comprehension

```
numeros = [5, 12, 20, 27]
print(numeros)

dobros = [x*2 for x in numeros]
print(dobros)

pares = [x for x in numeros if x % 2 == 0]
print(pares)

# criar uma lista com os ímpares convertidos para o par seguinte
imp_conv = [x+1 for x in numeros if x % 2 == 1]
print(imp_conv)
```

Variáveis Globais

Python - Global Variables

‹ Previous Next ›

Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

Example

Get your own Python Server

Create a variable outside of a function, and use it inside the function

```
x = "awesome"

def myfunc():
  print("Python is " + x)

myfunc()
```

Try it Yourself »

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

https://www.w3schools.com/python/python_variables_global.asp

Arquivos Texto

- A programação em arquivos de dados é uma habilidade essencial no desenvolvimento bem-sucedido de aplicações.
- Operações relacionadas a gravação e recuperação de dados armazenados em arquivos estão entre as mais importantes de qualquer linguagem de programação.
- Arquivos texto são utilizados para armazenar diversos tipos de informações, desde logs de ações realizadas em sistemas, até dados complexos formatados para a transferência de dados entre bancos.
- Um arquivo texto está projetado para ser lido do início até o fim toda a vez que for aberto.

Operações sobre Arquivos em Python

```
arq = open("acessos.txt", "r")   # abre o arquivo e associa ele a variável arq.
                                 # O 2º parâmetro indica o modo de abertura:
                                 # "w": criação; "r": leitura; "a": adição de dados
arq.write("usuário 2")           # escreve o texto no arquivo
tudo = arq.read(n)               # lê 'n' caracteres do arquivos.
                                 # Sem 'n' todo o conteúdo de arq é lido
linha = arq.readline( )          # lê uma linha do arquivo e posiciona na linha seguinte
linhas = arq.readlines( )        # lê todo o conteúdo do arquivo e joga em um vetor (linhas)
arq.close( )                     # fecha o arquivo
os.path.isfile("nomearq.txt")    # verifica se o arquivo existe (necessita de import os)
```

A segunda maneira de fechar um arquivo é usar a `with`instrução :

Python

```
with open('dog_breeds.txt') as reader:
    # Further file processing goes here
```

A `with`instrução se encarrega automaticamente de fechar o arquivo assim que ele sai do `with`bloco, mesmo em casos de erro. Eu recomendo fortemente que você use a `with`instrução o máximo possível, pois ela permite um código mais limpo e facilita o tratamento de erros inesperados.

Provavelmente, você também desejará usar o segundo argumento posicional, `mode`. Este argumento é uma string que contém vários caracteres para representar como você deseja abrir o arquivo. O padrão e mais comum é `'r'`, que representa a abertura do arquivo no modo somente leitura como um arquivo de texto:

Python

```
with open('dog_breeds.txt', 'r') as reader:
    # Further file processing goes here
```

https://realpython.com/read-write-files-python/

Salvar dados em arquivo texto

```python
def salva_dados():
    # modos: "r"=>ler, "a"=>adicionar, "w"=>escrever(recria o arquivo)
    with open("alunos.txt", "w") as arq:
        for nome, curso, parcela in zip(nomes, cursos, parcelas):
            arq.write(f"{nome};{curso};{parcela}\n")
```

Ler dados de arquivo texto

```
def carrega_dados():
  # se não existe o arquivo
  if not os.path.isfile("alunos.txt"):
    return

  with open("alunos.txt", "r") as arq:
    dados = arq.readlines()    # lê todas as linhas do arquivo
                               # e cria um vetor (de linhas)

    # percorre as linhas e acrescenta aos vetores
    for linha in dados:
      partes = linha.split(";")
      nomes.append(partes[0])
      cursos.append(partes[1])
      parcelas.append(float(partes[2]))
```

Usando `zip()` em Python

A função do Python `zip()` é definida como `zip(*iterables)`. A função recebe iteráveis como argumentos e retorna um **iterador** . Este iterador gera uma série de tuplas contendo elementos de cada iterável. `zip()` pode aceitar qualquer tipo de iterável, como arquivos , listas, tuplas , dicionários , conjuntos e assim por diante.

Passando `n` argumentos

Se você usar `zip()` with `n` argumentos, então a função retornará um iterador que gera tuplas de comprimento `n`. Para ver isso em ação, dê uma olhada no seguinte bloco de código:

Pitão

```
>>> numbers = [1, 2, 3]
>>> letters = ['a', 'b', 'c']
>>> zipped = zip(numbers, letters)
>>> zipped  # Holds an iterator object
<zip object at 0x7fa4831153c8>
>>> type(zipped)
<class 'zip'>
>>> list(zipped)
[(1, 'a'), (2, 'b'), (3, 'c')]
```

https://realpython.com/python-zip-function/

Ordenar unindo 2 listas: sorted e zip

```
>>> letras = ['b', 'a', 'c']
>>> numeros = [3, 2, 1]
>>> juntas = sorted(zip(letras, numeros))
>>> print(juntas)
[('a', 2), ('b', 3), ('c', 1)]
```

Separar novamente (unzip)

```
>>> letras2, numeros2 = zip(*juntas)
>>> letras2
('a', 'b', 'c')
>>> numeros2
(2, 3, 1)
```

Uso da função enumerate()

Iterando Sobre uma Lista

Um dos usos mais comuns da função `enumerate` é iterar sobre uma lista, obtendo tanto o índice quanto o valor de cada elemento.

```
1 alunos = ['Ana', 'Bruno', 'Carlos']
2 for indice, aluno in enumerate(alunos):
3     print(f'O aluno {aluno} está na posição {indice}')
```

Testar

Saída:

```
1 O aluno Ana está na posição 0
2 O aluno Bruno está na posição 1
3 O aluno Carlos está na posição 2
```

Exercícios: List Comprehension

1. Dada uma lista de números, crie uma segunda lista apenas com os números da lista original que são divisíveis por 10.
   - numeros = [15, 30, 50, 72, 95]
   - Numeros2 = [30, 50]

2. Dada uma lista de nomes, crie novas listas com: a) os nomes em maiúsculas; b) os nomes com apenas as primeiras letras do nome em maiúsculas.
   - nomes = ["ana júlia", "joão antônio", "luis eduardo", "maria helena"]
   - nomes2 = ["ANA JÚLIA", "JOÃO ANTÔNIO", "LUIS EDUARDO", "MARIA HELENA"]
   - nomes3 = ["Ana Júlia", "João Antônio", "Luis Eduardo", "Maria Helena"]

3. Dada uma lista de números, crie uma segunda lista apenas com raiz quadrada dos números da lista original que possuem raiz quadrada exata
   - numeros = [10, 16, 20, 25, 36, 40]
   - numeros2 = [4, 5, 6]